TOWARDS HYBRID CONTROL-FLOW AND DATAFLOW ARCHITECTURES

Nenad Korolija, PhD[1]
School of Electrical Engineering, University of Belgrade, Serbia

Vladisav Jelisavčić, PhD
Mathematical Institute of the Serbian Academy of Sciences and Arts, Serbia

Zlatogor Minchev, PhD
Institute of Information and Communication Technologies,
Bulgarian Academy of Sciences, Sofia, Bulgaria

Veljko Milutinović, PhD
Department of Computer Science,
University of Indiana in Bloomington, Indiana, USA

PURPOSE

According to Moore's law, number of transistors per integrated circuit doubles about every two years (Moore, 1998). This is predominantly achieved by making transistors smaller and smaller through advances in photolithography. However, the number of transistors per integrated circuit was not at its maximum at all times, but the density of transistors at which the cost per transistor was relatively close to the lowest. Namely, as the number of transistors increases, so does the probability of failure. It is well known that companies which were producing processors had even the possibility to reduce number of active cores if the production of any of cores went wrong.

For decades, the increase in frequencies of processors was almost following the Moore's law, by doubling about every two years. Unlike it is the case with increasing the number of transistors per unit measure of a surface, increasing the frequencies also increases the power consumption and therefore heating as well. It can be roughly estimated that the power consumption is proportional to the

1 nenadko@etf.bg.ac.rs

square of the integrated circuit frequency. More importantly, the increased power consumption leads to the higher cooling requirements. As a result, recent generations of central processing units (CPUs) are usually limited around 3GHz, while the number of cores keeps growing.

High performance computing (HPC) benefits from increased number of cores of today's CPUs. However, in order for an algorithm to benefit from relatively high number of available cores, the algorithm has to be scalable enough. The scalability is limited not only by properties of an algorithm, but by the architecture it is executed on. For example, by increasing the communication time between two cores, the algorithm benefits less from using multiple cores in parallel. In order to justify the usage of another core, the algorithm has to run faster by utilizing the core than it otherwise would.

Control-flow architectures, also referred to as von Neumann architectures, define the temporal sequence of individual instructions of a computer algorithm. As such, they are complex enough to be able to execute all instructions defined by the architecture. On the other hand, they can execute only a single instruction without multiplying control-flow computational units or dividing them onto instruction phases they are responsible for. One way of parallelizing instruction execution is using the pipeline. Multicore computer architectures consist of multiple control-flow type of processors that further increase the number of instructions that can be run in parallel.

So-called manycore architectures are based on control-flow principles, but the number of processing elements they include is usually couple of orders of magnitude greater than of CPUs. Processing elements are usually simpler and suitable for scalable algorithms. The primary purpose of manycore architectures was to support the necessity for fast processing in order to render screen frames in graphics demanding applications. Later, their capability of executing multiple instructions in parallel was utilized for parallelizing and therefore accelerating many control-flow applications.

Unlike it is the case with control-flow architectures, dataflow architectures are configured to execute a single algorithm before they are reconfigured for a new purpose (Trifunovic, Milutinovic, Salom & Kos, 2015). Reconfigurability is usually achieved using the Field-programmable gate array (FPGA). In the case of dataflow architectures, data literally flows through the hardware, producing results at the output based on the data received at the input. There are many advantages of this approach. For example, processing element responsible for executing a single instruction can be configured to be able to execute only that instruction. As such, processing element is less complex comparing to the control-flow architecture, and therefore smaller. Another benefit is that the data travels in parallel from any processing element producing semi-result to those that consume the result. Data-

flow architectures are suitable for executing algorithms that include considerable amount of repetition.

With raising number of transistors per chip, one could argument for combining multiple computing architectures at the same chip die, forming a hybrid architecture. Some of the benefits include reduced hardware size comparing to the size of multiple computer architectures separately, reduced power consumption, but, more importantly, improvements in communication speed between control-flow and dataflow hardware.

Multiple computer architecture paradigms available in a single desktop computer or server node impose relatively slow communication speed due to the distance between them. In addition, powering multiple architectures might require more efforts in terms of cooling comparing to the cooling of a single chip.

A hybrid control-flow and dataflow architecture on a single chip might solve these issues, enabling certain algorithms to be transformed and programmed for the execution on merged control-flow and dataflow hardware simultaneously.

Many high performance algorithms perform certain operations on a matrix of data, where edge elements are processed differently than the rest of the matrix. Auxiliary processing might be required between two consecutive processing of a matrix, where the amount of processing would be neglectable comparing to the processing of a matrix. There are few possibilities to implement this using the dataflow paradigm. One is to create separate kernels for processing middle matrix elements and edge elements, and even another one for processing corner elements. However, due to the complexity of orchestrating streams of the input data, leading them to appropriate kernels, as well as collecting results from the output of kernels, one could implement a single kernel to handle all the cases, where conditionals would be used to cope with the differences in processing. Similarly, separate kernels could handle processing of vectors between two consecutive processing of matrices, or a single kernel could be extended to handle this scenario as well. In any case, the complexity of dataflow hardware grows reasonably (e.g. could be increased more than twice), where the execution time of the CPU needed for processing edge elements and auxiliary processing is less than 1% of the total execution time that a CPU will need to process the whole algorithm. Dataflow implementation of the Lattice-Boltzmann algorithm is one example application that matches the previous scenario (Korolija, Djukic, Milutinovic & Filipovic, 2013).

The goal of this research is to exploit the possibility of merging dataflow and control-flow paradigms on the same chip die with communication speed matching those of cache memories, and to show on the example of the Lattice-Boltzmann algorithm the potential benefits.

Further sections describe control-flow, multicore, manycore, and dataflow architectures in more detail, and explain the proposed hybrid architecture. The presentation is based on the proposed method (Milutinovic, 1996), and is reviewed by authors in accordance with the proposed activity diagram (Banković et al., 2020).

DESIGN/METHODS/APPROACH

This section sheds some light on available computing paradigms used in high performance computing. Therefore, although control-flow single-core processors have been replaced by multicore processors since relatively long time ago, the paradigm will be explained, as most computer clusters and computer clouds include processors based on the control-flow.

Control-flow computer architectures are based on the principles defined by John von Neumann. First control-flow processors consisted only of a single-core, being capable of executing one instruction at any particular moment. This imposes relatively high complexity for a processor that can execute only one instruction at any given moment. The utilization of transistors raised with the introduction of pipelines that enabled executing few instructions simultaneously, but in different stages – while one instruction was being fetched, another one is being executed, and another one is writing results, etc. The improvement in the capacity of executing many instructions per second has been improving predominantly due to the constant raise of frequencies the processors operated at, approximately doubling each second year. Due to the relatively high increase in power consumption with raising frequencies, multicore processors appeared as a logical consequence, enabling faster execution at a smaller price.

Multicore computer architectures are based on the same principles as control-flow computer architectures. Unlike first single-core architectures, multicore architectures include multiple cores, where each core can execute multiple instructions simultaneously. Advances in technology of producing processors led to reducing transistor sizes. As a result, more CPUs could have fit within the same chip die. However, multicore computers suffer from the same problems as conventional single-core computers. The internal bus can become the bottleneck, as many instructions that run in parallel on a single-core, but in different phases of instruction execution, try to write results back to e.g. a cache memory or registers and to read from the memory/registers simultaneously. The number of instructions is limited by the architecture, allowing running only few instructions per core in parallel.

Advances in computer graphics in recent decades lead to defining a term many-core used for control-flow type of processing on graphics card that can include

thousands of processing units. Manycore computer architectures, or so-called graphics processing unit (GPU) processors, are based on the same principles as other control-flow processors. However, unlike it is the case with single-core and multicore processors, manycore processors can include thousands of processors. As a result, they are highly utilized for parallelizing execution of highly scalable computing algorithms. The lack of these architectures is that they are usually simpler than conventional multicore processors. Therefore, they cannot execute all instructions defined by the multicore processors. It is worth saying that computer architectures that include GPUs usually include also multicore processors, where a multicore processor is responsible for initializing the data for GPUs, starting the processing, and collecting results, while GPUs are utilized for the portion of algorithm or algorithms that are highly scalable.

Dataflow computing paradigm is based on the data flowing through the hardware (Trifunovic et al., 2015). Fig. 1 depicts an example processing implemented using the dataflow paradigm. Function elements from the figure can be observed as processing elements that are mutually connected. The memory could either be placed on the same chip with the dataflow hardware, or on a CPU that is used along with the dataflow hardware, or as a separate unit. The input data would be streamed into the dataflow hardware, and results of the execution would be streamed back to the CPU in order to be further processed or stored in the main memory.

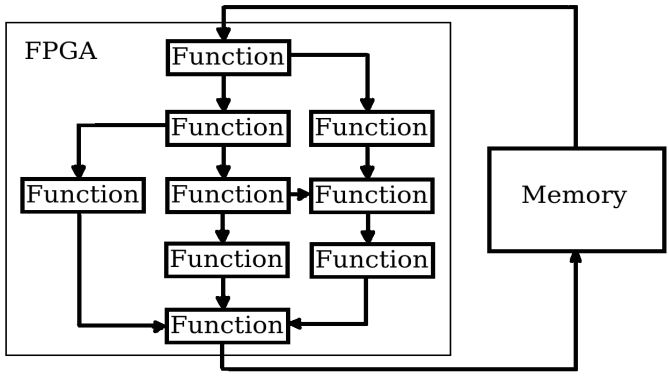

Figure 1: *Dataflow programming model.*

Main benefits of dataflow hardware comparing to the control-flow hardware are the following:

- Data produced at a processing element travels directly to those processing elements that need this data. This way, many data transfers can be processed in parallel. This solves the previously mentioned problem of a bus as a bottleneck of the control-flow type of architecture.

- Processing elements are simpler than conventional control-flow type of processors, as they are utilized for a single instruction. This makes them smaller than conventional control-flow processors.
- Higher density in processing elements comparing to the manycore enables more processing elements to be placed on the same chip die. Besides that, it also affects the performance of algorithm execution directly, as the time needed to send the data from one processing element to another is nearly proportional to the distance between them.

The algorithm execution is performed using the dataflow hardware in the following steps:

- Control-flow processor prepares the data to be processed by the dataflow kernels.
- Control-flow processor initializes the dataflow hardware. In some cases, data is streamed to the dataflow kernels for processing. In other cases, the data is copied into the dataflow hardware memory, so that the data can be processed faster by kernels.
- Dataflow hardware processes the data, streaming the output back to the CPU, or streaming the results into the dataflow hardware memory that can be later read by the CPU, or another dataflow hardware kernel.
- Control-flow processor collects results of executing the dataflow hardware and processes it further.

Dataflow architectures are capable of accelerating many high performance computing algorithms (Milutinović, Furht, Obradović & Korolija, 2016). Examples include, but are not limited to, sorting (Kos, Ranković, & Tomažič, 2015) and providing biofeedback in sport (Umek & Kos, 2016).

However, compared to the control-flow applications, developing dataflow applications usually requires more time due to the constraints imposed by the dataflow hardware. These constraints lead to considerably higher programmer task-type efforts (Popovic, Bojic & Korolija, 2015), especially with non-functional requirements (Popović, Korolija, Marković, & Bojić, 2017).

Application-specific integrated circuit (ASIC) is an integrated circuit (IC) chip capable of executing on a hardware the algorithm that the hardware is designed for. Algorithms for ASIC can usually be implemented using the dataflow programming model. As a result, ASIC is expected to be highly optimized, allowing many instructions to be executed in parallel. Example include Google Cloud Tensor Processing Units (TPUs) used for machine learning algorithms. This kind of architecture can exist in a cluster or cloud, where one could expect a range of algorithms utilizing the dedicated hardware, but it is not that common in desktop

computers, as it is hard to justify why an ordinary user would have the need for a specialized hardware for multiple algorithms.

In contrast to ASIC, dataflow architectures based on FPGAs can be reconfigured, so that they can support implementation of many algorithms using the dataflow paradigm. One of the biggest challenges in developing algorithms for dataflow architectures is the configuring of FPGAs. Researchers have developed a method for automatically translating algorithms from the control-flow to dataflow paradigm (Milutinovic et al., 2017; Korolija, Popović, Cvetanović & Bojović, 2017). Although the dataflow paradigm exists since 1960s, programmers are usually specialized in programming control-flow architectures. Most of the computer programs are written for computer architectures based on control-flow. The same applies to the development of compilers and frameworks. Authors of this manuscript believe that recent advances in dataflow architectures along with the raise of the need for machine learning algorithms will lead to the increase in usage of dataflow architectures and the percentage of programming community educated for dataflow programming.

HPC solutions can include a wide range of building blocks, from custom motherboard designs, to system configurations, rack designs along with cooling systems, etc. At the beginning of HPC, most HPC solutions were built for general purpose compute intensive workloads, and were used for various purposes such as digital manufacturing, banking services, medical research, oil and gas production, etc. Today's HPC architectures can include relatively high amount of racks filling a big room with multiple nodes per rack (e.g. around 100) and thousands of processor cores per rack. In terms of computing paradigms, HPC architectures may include dataflow cards attached on some or all nodes, but they almost always include control-flow type of processors, including multicore processors and, optionally, manycore processors.

For already decades, the frequencies of processors are not doubling any more. Instead, the transistor count per chip die is increasing, which allows putting more and more cores onto a single chip die. Authors of this manuscript believe that, in the near future, it will be beneficial for the processor to include not only few cores, but in addition manycore architecture and dataflow architecture. Such a hybrid processor can have higher communication speed between the control-flow and the dataflow hardware, since they could share the same cache, or communicate directly using the internal bus. The chip could also include Network on chip (NoC), ethernet support, and appropriate control logic. Fig. 2 depicts a hybrid control-flow and dataflow computer architecture on a single chip die.

The idea of combining control-flow and dataflow hardware on the same chip die is not new (Yazdanpanah, Alvarez-Martinez, Jimenez-Gonzalez & Etsion, 2013). Researchers have achieved an average speedup factor of 3 to 11 over an NVIDIA GPGPU, while having better energy efficiency (Voitsechov & Etsion, 2015). There

is a research that proposes including also Internet of things on the same chip die (Milutinović et al., 2021). Efforts on combining control-flow and dataflow architectures can be roughly divided onto:

- Solutions including control-flow hardware with a software dataflow;
- Control-flow hardware controlling distant dataflow hardware usually accessed via PCIe bus.

One of the key benefits of the proposed architecture is that it allows defining computer architecture in the run time by defining a set of instructions that the architecture will be capable of executing, where these instructions could be executed much faster than it would be possible on a conventional CPU. For example, one could configure FPGAs for certain instructions, and automatically generate the compiler for such an architecture, where instructions supported by the dataflow hardware would execute on the dataflow hardware, while other instructions would be executed using one of the CPUs. This feature is not explored in more detail in this manuscript.

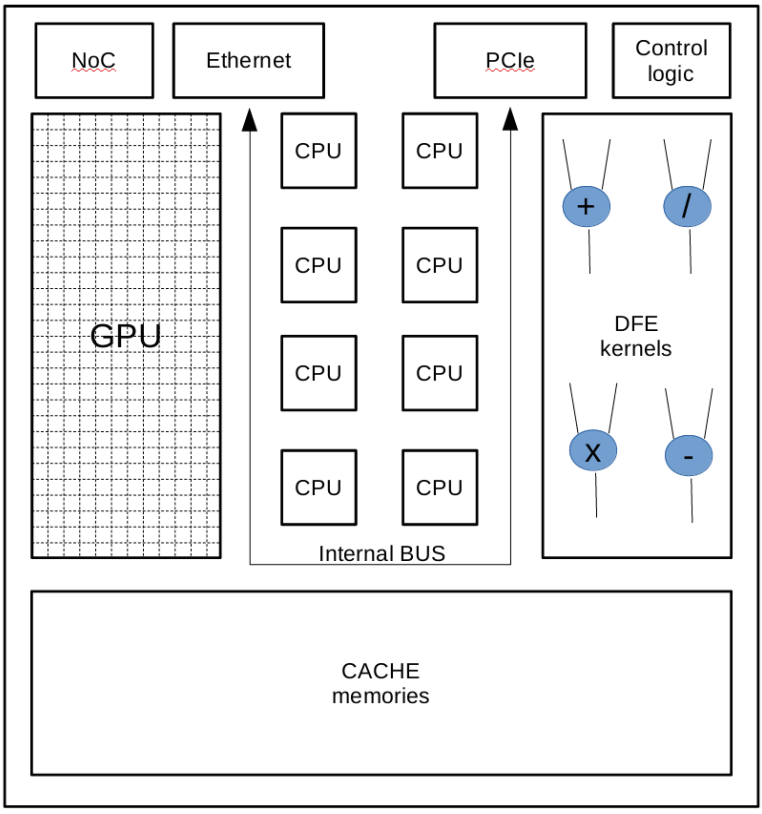

Figure 2: *Hybrid control-flow and dataflow computer architecture.*

Control logic would need to be much more sophisticated comparing to the existing to support cache sharing across heterogeneous architectures. Similar problem was already tackled with the research that, to some extent, splits temporal and spatial cache data (Sustran, Rakocevic, Milutinovic, 2015), proving that such a concept is implementable.

FINDINGS

The analysis of the potentials for accelerating algorithms using the proposed hybrid computer architecture is divided into four stages. First, the communication speed between the control-flow and the dataflow hardware is calculated and compared to the speed of PCIe bus. Second, the benchmark is defined by analyzing open source implementations of dataflow algorithms for the same dataflow hardware type. Third, the simulation analysis is performed assuming various ratios of jobs from the benchmark. Fourth, the comparison between the proposed hybrid architecture and existing architectures is performed.

The communication speed of the PCIe 6.0 is 128GB/s, where the latency is less than 10ns. If the hybrid architecture has similar cache to the one of processor i9-9900k, running at 3.6GHZ, the L3 cache speed would be 300GB/s with the latency around 11ns, and the L1 cache speed would be around 3TB/s with the latency of 0.8ns. The speed of communication with the main memory is 47GB/s with the latency of 45ns. In the most rigorous approach, one could expect the dataflow hardware to be connected with the manycore architecture over the L3 cache (not L1 or L2), resulting in the communication acceleration factor of 2,34375 comparing to the PCIe 6.0.

In order to provide a fair comparison, first four dataflow applications from the Application gallery available in the open literature (Trifunovic, Milutinovic, Korolija & Gaydadjiev, 2016), for which there were enough data to estimate the possibility for acceleration using the proposed hybrid architecture, are chosen.

First application is Huxley muscle model with claimed acceleration factor of around 33. As the problem size is not given for this application, the worst case is considered, which includes neglectable amount of edge elements comparing to the total number of elements to be processed. Therefore, assuming that the edge elements can be calculated using the manycore architecture, the middle elements can be calculated using simpler kernels. Current dataflow implementation of the Huxley muscle model includes four conditional arithmetical statements out of the total of 14 arithmetical statements that include these four as well. CPU execution time varies between 5s and 60s. For the four conditional arithmetical statements, it is estimated that they can be accelerated by the factor of two in the case there is

no need to check for boundary conditions. It is also assumed that the complexity of arithmetical statements doesn't vary. The total acceleration possibility is roughly calculated as a reduction from 14 to 12, i.e. the acceleration factor is around 1,167. This is the worst case application, as boundary elements are not included into the calculation, neither the speed of the communication between the control-flow and the dataflow hardware.

The second application is Poisson solver. It can be measured only for 32x32x32 space as input, because for greater input design cannot fit. The acceleration factor is claimed to be around 124. From the total input space, there are 32*32*6 - 8*2 edge elements, which is 18.7%. If we run 18.7% of calculations using the manycore architecture, and (1–0,187)% using the dataflow architecture, we will achieve the acceleration factor of 1.23, assuming that manycore architecture can process these 18.7% of data in parallel with dataflow hardware processing the rest of the data.

The third application is Simplex. The bottleneck for both implementations is the bandwidth between the control-flow and the dataflow hardware. The acceleration factor is claimed to be around 5. According to the acceleration graph based on the problem size, the steep curve flattens at about 5, resulting in the limited acceleration for problems of sizes greater than 10 to the power of four. If the communication speed is 2.34 times faster, the resulting acceleration factor would be 11.7, while the relative acceleration introduced by the proposed hybrid architecture would be 2.34.

The fourth dataflow application is Network sorting. The acceleration factor is claimed to be in the ranges between 7 and 12, if the sorting times include communication delays between the control-flow and the dataflow hardware, and between 100 and 160 when considering only the sorting time inside the FPGA. If we consider the mathematical average accelerations of 9.5 and 130 for both scenarios respectively, the difference of 120.5 corresponds to the communication. If we multiply the acceleration with the communication included by 2.34, this will result in 22.2 acceleration factor with the communication included. Compared to the acceleration factor of 130 in the case that no communication is performed, one can conclude that even with the proposed hybrid model the communication speed would still be the bottleneck.

In average, the acceleration factor for all dataflow applications from the benchmark is around 1.77. By tuning the ratio between dataflow jobs, it can reach close to the 2.34, or drop to 1.17. However, there is a threat to validity due to relatively small subset of dataflow applications that are used in the evaluation of the proposed hybrid architecture, but the simple analysis reveals that there are potentials to accelerate the dataflow application, whose bottleneck is in the communication between the control-flow and dataflow hardware, by the factor of 2.34 or even more.

Some of the potentials of combining control-flow and dataflow architectures are already known (Milutinović, Trifunović, Korolija, Popović & Bojić, 2017). Authors have also been working on recovering network connectivity structure, developing a very fast Scale-Free Networks Estimation Through Cholesky factorization (SNETCH) optimization algorithm based on coordinate descent. This highly parallelizable algorithm is suitable for dataflow architectures. The topological problem it solves can be matched in detecting crimes (Jelisavcic, Stojkovic, Milutinovic & Obradovic, 2018). Dataflow processing can improve video surveillance in terms of better real-time face recognition (Bhowmik, Garcia, Wallace, Stewart & Michaelson, 2017), but also in terms of tracking subjects based on video streams from multiple locations, reducing the total power consumption at the same time.

Another advantage of the proposed hybrid architecture compared to the typical dataflow architectures that are connected to CPUs is in the fact that the dataflow hardware can be better utilized, since only those parts of algorithms that are executed repeatedly again could be executed using the dataflow hardware, while instructions that are not that often executed could be executed using the many-core. As an example, the Lattice-Boltzmann algorithm predominantly calculates matrix element values. The processing differs for the elements in four corners of the matrix from those belonging to edges of the matrix, and the rest of elements require different processing. Although this processing doesn't differ much, and can be achieved using a single dataflow kernel, this kernel has higher complexity and more electrical power and time is required for processing a single value.

When it comes to using the proposed hybrid architecture in a computer cluster, one can think of dividing the matrix by X and Y coordinates onto multiple hybrid processors that would be responsible each for their own domain of data. As a result, edge and corner elements would have to be exchanged with neighborhood processors so that the time needed for processing these elements and the communication with neighborhood processors differs by orders of magnitude from the time needed to process matrix elements that can be processed independently from other processors.

The additional constraint to using the proposed hybrid architecture in clusters is scheduling for hybrid control-flow and dataflow architectures. Researchers have developed relatively fast scheduling algorithms comparing to the usual duration time of dataflow jobs (Korolija, Bojic, Hurson & Milutinovic, 2022).

One important aspect of the combined control-flow and dataflow hybrid processor is the expected lifetime and counterfeit detection. The duration is expected to be around the shortest lasting from the three incorporated architectures, and the possible counterfeit hybrid processor chips could be identified using existing statistical methods (Huang, Liu, Korolija, Carulli & Makris, 2015).

ORIGINALITY/VALUE

This manuscript presents relevant aspects of control-flow and dataflow paradigms for running scalable algorithms in parallel, and advocates for a hybrid hardware available on the same chip die.

The control-flow computing paradigm assumes that the computer architecture is able to execute any of the instructions defined by the instruction set at any given moment. This imposes that the hardware must be relatively complex comparing to the one needed for executing a single instruction. Modern trend is increasing the number of processing units within a central processor or graphics card, often referred to as multicore or manycore GPUs, respectively. This increases the parallelism of scalable algorithms, but the number of transistors divided by number of instructions that can run in parallel is still relatively high.

Dataflow paradigm reduces this number by an order of magnitude or more by allowing data to flow through the hardware. The downsides of using the dataflow paradigm are that one needs to configure the dataflow hardware and that the control-flow type of processor is usually needed for initialization of the data, starting the dataflow hardware, and collecting results. Communication lag between control-flow and dataflow hardware reduces the possibility of parallelizing certain algorithms that have low computation to communication ratio.

Hybrid multicore, manycore, and dataflow computer architecture can fit within a single chip. As a result, the communication speed drops comparing to the communication between a processor and a distant dataflow hardware. This offers new possibilities in terms of algorithms that can be accelerated using both paradigms, but also in terms of execution times of algorithms for the dataflow architectures. Authors believe that this can improve important aspects of modular technical system modelling and real-time analysis of complex signals obtained from multiple locations (Minchev and Atanassov 2005; Popivanov et al., 2006).
The idea of combining multiple computing paradigms in order to accelerate high performance computing algorithms is not new, but the presented work sheds light on one possible implementation of the chip that includes both multicore, manycore, and dataflow architecture. Based on real problems that are implemented for dataflow paradigm using a single framework, and the appropriate scheduling algorithm, the presented approach proved to have potentials to accelerate certain high performance algorithms by factors higher than two, reducing the energy consumption at the same time.

ABBREVIATIONS:

CPU – Central processing units

HPC – High performance computing

FPGA – Field-programmable gate array

GPU – Graphics processing unit

ASIC – Application-specific integrated circuit

IC – Integrated circuit

TPU – Tensor Processing Unit

NoC – Network on chip

SNETCH – Scale-Free Networks Estimation Through Cholesky factorization

REFERENCES

Moore, G. E. (1998). Cramming more components onto integrated circuits. Proceedings of the IEEE, 86(1), 82–85.

Trifunovic, N., Milutinovic, V., Salom, J., & Kos, A. (2015). Paradigm shift in big data supercomputing: dataflow vs. controlflow. Journal of Big Data, 2(1), 1–9.

Korolija, N., Djukic, T., Milutinovic, V., & Filipovic, N. (2013). Accelerating Lattice-Boltzman method using Maxeler dataflow approach. The IPSI BgD Transactions on Internet Research, 34.

Milutinovic, V. (1996). The best method for presentation of research results. IEEE TCCA Newsletter, (9), 1–6.

Banković, M., Filipović, V., Graovac, J., Hadži-Purić, J., Hurson, A. R., Kartelj, A., ... & Živković, M. (2020). Teaching graduate students how to review research articles and respond to reviewer comments. In Advances in Computers (Vol. 116, No. 1, pp. 1–63). Elsevier.

Milutinović, V., Furht, B., Obradović, Z., & Korolija, N. (2016). Advances in high performance computing and related issues. Mathematical problems in engineering, 2016.

Kos, A., Ranković, V., & Tomažič, S. (2015). Sorting networks on Maxeler dataflow supercomputing systems. In Advances in Computers (Vol. 96, pp. 139–186). Elsevier.

Umek, A., & Kos, A. (2016). The role of high performance computing and communication for real-time biofeedback in sport. Mathematical problems in engineering, 2016.

Popovic, J., Bojic, D., & Korolija, N. (2015). Analysis of task effort estimation accuracy based on use case point size. IET Software, 9(6), 166-173.

Popović, J., Korolija, N., Marković, Ž., & Bojić, D. (2017, November). The influence of non-functional requirements in UCP method on the accuracy of effort estimates. In 2017 25th Telecommunication Forum (TELFOR) (pp. 1–4). IEEE.

Milutinovic, V., Salom, J., Veljovic, D., Korolija, N., Markovic, D., & Petrovic, L. (2017). Transforming applications from the control flow to the dataflow paradigm. In Dataflow supercomputing essentials (pp. 107–129). Springer, Cham.

Korolija, N., Popović, J., Cvetanović, M., & Bojović, M. (2017). Dataflow-based parallelization of control-flow algorithms. In Advances in computers (Vol. 104, pp. 73–124). Elsevier.

Yazdanpanah, F., Alvarez-Martinez, C., Jimenez-Gonzalez, D., & Etsion, Y. (2013). Hybrid dataflow/von-Neumann architectures. IEEE Transactions on Parallel and Distributed Systems, 25(6), 1489–1509.

Voitsechov, D., & Etsion, Y. (2015, December). Control flow coalescing on a hybrid dataflow/von Neumann GPGPU. In Proceedings of the 48th International Symposium on Microarchitecture (pp. 216–227).

Milutinović, V., Azer, E. S., Yoshimoto, K., Klimeck, G., Djordjevic, M., Kotlar, M., ... & Ratkovic, I. (2021, June). The ultimate dataflow for ultimate supercomputers-on-a-chip, for scientific computing, geo physics, complex mathematics, and information processing. In 2021 10th Mediterranean Conference on Embedded Computing (MECO) (pp. 1–6). IEEE.

Sustran, Z., Rakocevic, G., & Milutinovic, V. (2015). Dual Data Cache Systems: Architecture and Analysis. Advances in Computers, 96.

Trifunovic, N., Milutinovic, V., Korolija, N., & Gaydadjiev, G. (2016). An AppGallery for dataflow computing. Journal of Big Data, 3(1), 1–30.

Milutinović, V., Trifunović, N., Korolija, N., Popović, J., & Bojić, D. (2017, November). Accelerating program execution using hybrid control flow and dataflow architectures. In 2017 25th Telecommunication Forum (TELFOR) (pp. 1–4). IEEE.

Jelisavcic, V., Stojkovic, I., Milutinovic, V., & Obradovic, Z. (2018). Fast learning of scale-free networks based on Cholesky factorization. International Journal of Intelligent Systems, 33(6), 1322–1339.

Bhowmik, D., Garcia, P., Wallace, A., Stewart, R., & Michaelson, G. (2017). Power efficient dataflow design for a heterogeneous smart camera architecture.

Korolija, N., Bojic, D., Hurson, A. R., & Milutinovic, V. (2022). A runtime job scheduling algorithm for cluster architectures with dataflow accelerators. Advances in Computers, 201.

Huang, K., Liu, Y., Korolija, N., Carulli, J. M., & Makris, Y. (2015). Recycled IC detection based on statistical methods. IEEE transactions on computer-aided design of integrated circuits and systems, 34(6), 947–960.

Popivanov, D., Stomonyakov, V., Minchev, Z., Jivkova, S., Dojnov, P., Jivkov, S., Christova, E., & Kosev, S. (2006). Multifractality of decomposed EEG during imaginary and real visual-motor tracking. Biological Cybernetics, 94(2), pp. 149–156.

Minchev, Z., & Atanassov, K. (2005). On the possibility for generalized nets modelling of modular robotic system. Advanced Studies on Contemporary Mathematics, 10(2), pp. 169–174.